OBJECT ORIENTED DATA STORE INTEGRATION ENVIRONMENT FOR INTEGRATION OF OBJECT ORIENTED DATABASES AND NON-OBJECT ORIENTED DATA FACILITIES

FIELD OF THE INVENTION

The present invention relates to computer databases, and more particularly to integration of object oriented databases and non-object oriented data facilities.

BACKGROUND OF THE INVENTION

Various means and methodologies exist presently for persistent storage of data for use in computer system applications. Known database management systems (DBMS's) facilitate storage of data in non-volatile storage, e.g. disks, tapes, etc, for use even after the program that used or generated the data is terminated ("persistence"). In addition to persistence, DBMS's fundamentally provide "concurrency control" so users of data can share a database without interfering with each other or compromising the integrity of data, and "recovery" features to protect and restore data integrity upon system (hardware or software) failures. DBMS "query" facilities enable users to access the large volumes of data within a database by specifying some particular characteristic or field within data records, while "security" features are typically built into DBMS's to limit access to some data. Known DBMS's support "schema management" for describing properties of, and relationships between, data in a database.

With the evolution of DBMS technology, fundamental functionality as described hereinbefore has been maintained and expanded while data complexity and processing performance requirements have increased. A type of data management technology applied in commercial data processing known as "relational" DBMS, is modeled to be relatively simplistic in that all data is organized as though it is formatted into tables, with the table columns representing the table's fields or domains and the table rows representing the values of the table's fields or domains. Data is logically organized as tables but is not necessarily physically stored as such. The relational database user does not need to know how the database is physically constructed and can access and update data via a language interface or "structured query language" (SQL).

The relative complexity of data associated with science and engineering problem solving, and the evolution of complex data structures and data entities modeled on real-world objects led to the development of a new generation of DBM's known as "object DBM's" (ODBM's) or "object oriented DBM's" (OODBM's). ODBM's do not conform to the relational model but provide virtually the same fundamental functionality (i.e. persistence, concurrency control, recovery, security, query facilities and schema management) for storing and manipulating object entities. Object entities or "objects", are complex data structures which model real-world entities, and are associated in classes and identified with their informational features (attributes) and functional features (behaviors). Objects are effected using object oriented programming (OOP) languages such as C++ and Smalltalk. By defining complex, specialized data structures or objects that model real-world entities, program development is made easier and more natural as the level of abstraction of data is raised to a point where applications can

be implemented effectively in the same terms in which they are described by the users of the application. Objects are more readily classifiable into types, which are easily related to one another in subtype/supertype hierarchies. OOP languages permit the programmer to flexibly define data types so as not to be constrained by limited predefined types. OOP language types can be associated in classes which can "inherit" attributes and/or behaviors from other classes. Complex object data structures and types are not supported by the relational DBMS model, but by ODBM's which facilitate direct storage and manipulation of objects, without the need to map them into tables.

Relational DBM's and Object DBM's co-exist presently, with little likelihood that one will completely displace the other. Each type of DBMS (i.e. Relational and Object) is best suited for respective particular applications, e.g. Relational for the predefined data types of business data processing such as in the insurance and banking industries; and Object for the extensible data structures modeled on real-world entities such as used in computer-aided design and computer-aided software engineering. However, with the considerable investment associated with existing relational data stores, and the continuing evolution toward and appreciation for object applications, there exists a need for integrating object and relational technologies.

The need for integration of relational and object data typically will arise in situations where new object oriented applications are implemented to take advantage of aspects of object oriented programming in a context where data was/is managed according to the relational model. Known means of integrating object programming with a relational database include doing a manual, programming intensive conversion of all the relational data in the relational DBMS to object data, that is readily accessible to the object oriented programs. However, in addition to the significant efforts required for such conversion, there typically are non-object oriented application programs that continue to require access to the relational data. Thus, disadvantageously, it may be necessary to have redundant data stores resulting in duplicative resources and greater overhead. Additionally the task of updating and maintaining the Relational and Object versions of the DBMS creates difficulties in that updates must be substantially simultaneously coordinated and may have to be replicated in disparate environments. Even if the entire collection of existing programs that access the relational database are rewritten in an object oriented programming language, such a mode of conversion is an expensive, complicated and time consuming endeavor.

Standardized import/export facilities are known which permit importation of data in a predictable input format into an object database. The import/export facility is a program, implementation of which requires knowledge of the schema or format of the relational data. Additionally, the relational data elements must be mapped to objects within an ODBMS which are to be managed by the ODBMS which consequently manages the mapped relational data elements. Such facilities, however, lack flexibility in that a predefined format is required for representing data that is to be passed between relational and object environments. While the import/export facility effectively acts as a translation mechanism, there must be rigid adherence to the predefined format in which the relational data is maintained, otherwise it cannot be mapped to objects and managed by the ODBMS.

Translation techniques in the form of SQL Gateways are known which allow object language programs to retrieve relational data from a relational database in a form approximating objects rather than tables. A programmer must know